Programmieren mit FORTRAN

aus: *ComputerKurs*-Magazin, Jahrgänge 1984/1985

Frühe Formeln

Obwohl FORTRAN schon Mitte der fünfziger Jahre entwickelt wurde, hat sich die Sprache viele ihrer ursprünglichen Merkmale erhalten. Im ersten Artikel dieser neuen Serie sehen wir uns den allgemeinen Programmaufbau an und untersuchen, wie Ein- und Ausgabe und die mathematischen Abläufe funktionieren.

FORTRAN wird im allgemeinen als die erste Hochsprache angesehen. Die Sprache wurde in einer Zeit entwickelt, als Computer fast ausschließlich für die Zahlenverarbeitung eingesetzt wurden und kaum Möglichkeiten der Stringverarbeitung hatten. FORTRAN ist noch immer eine der wenigen Sprachen, die komplexe Zahlen direkt verarbeiten. Mit ihrer mathematischen Struktur lassen sich schnelle und leistungsfähige Routinen schreiben, die sich auch für Grafik eignen. Es gibt eine Reihe Maschinen, deren Betriebssysteme zumindest teilweise in FORTRAN geschrieben wurden.

Über Jahrzehnte entstand eine große Zahl Subroutinen für ein breites Spektrum numerischer Anwendungen. Dies ist einer der Hauptgründe für die Beliebtheit der inzwischen schon recht betagten Sprache. Da FORTRAN die einfachste Methode ist, diese Software einzusetzen, scheint sie noch für einige Zeit eines der wichtigsten Programmierinstrumente für Wissenschaftler und Ingenieure zu bleiben.

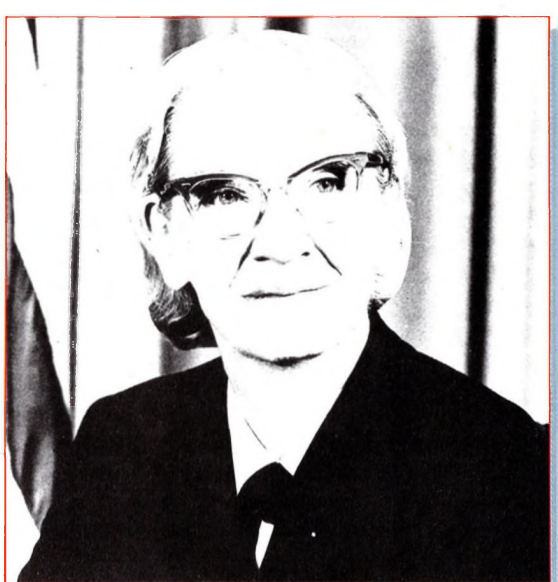
Auch für Microcomputer gibt es FORTRAN-Versionen, die jedoch nur selten den vollen Standard des aktuellen FORTRAN 77 erreichen. Die meisten sind Abwandlungen von FORTRAN IV mit entsprechenden Erweiterungen. Wir werden mit FORTRAN IV beginnen, da diese Version den besten Einblick in die Sprache bietet und später die Erweiterungen und Abweichungen vom Standard erläutern.

Der Aufbau von FORTRAN wird wesentlich von der Tatsache geprägt, daß früher für die meisten Ein- und Ausgaben 80-spaltige Lochkarten verwandt wurden. Die letzten acht Spalten einer Karte enthielten normalerweise die Kartennummer, so daß für jeden Befehl 72 Spalten zur Verfügung standen. FORTRAN kann daher pro Zeile nur einen Befehl mit einer Länge von maximal 72 Zeichen verstehen. Wenn ein Befehl sich über mehrere Zeilen erstrecken soll, müssen die folgenden Zeilen als Fortsetzung markiert sein. Die 72 Spalten werden noch weiter eingeschränkt: Die ersten fünf Spalten sind für die Nummer des Befehls reserviert, während die sechste anzeigt, ob eine Fortsetzungszeile vorliegt. Der eigentliche Befehl beginnt erst ab Spalte 7.

FORTRAN-Zeilen brauchen nicht wie in BASIC numeriert zu werden. Jede Zeile, die von einem anderen Befehl angesprochen wird, muß jedoch mit einer Zahl als Label versehen sein. Diese Nummern brauchen nicht in einer bestimmten Reihenfolge stehen, müssen aber eindeutig sein. Ein C in Spalte Eins markiert die Zeile als Kommentar. Weil Datentypen und Steuerstrukturen fehlen, sind FORTRAN-Programme schwer zu strukturieren. Da die Sprache schwer zu lesen ist, spielen Kommentare eine wichtige Rolle.

Moderne Zeiten

1954	Herausgabe der ersten FORTRAN-Beschreibung
1957	Erster Compiler für FORTRAN I
	Herausgabe der Beschreibung für FORTRAN II
1958	Erster Compiler für FORTRAN II
1962	Herausgabe der Beschreibung für FORTRAN IV
1963	Compiler für FORTRAN IV
1977	Herausgabe der Beschreibung für das neue FORTRAN 77
1978	Die ersten Compiler für FORTRAN 77



Grace Hopper
Grace Hopper war die treibende Kraft bei der Entwicklung der ersten Programmiersprachen. In den frühen fünfziger Jahren arbeitete sie bei Remington Rand und half bei der Entwicklung eines der ersten Compiler – A-2 – und den Sprachen ARITHMATIC und FLOW-MATIC. Die Firma IBM verfeinerte in ihrer ersten FORTRAN-Version das Konzept der compilierten Sprachen.

Wie in BASIC müssen Variablen nicht deklariert werden. Sie lassen sich an jedem Punkt des Programms einfach durch Angabe eines neuen Namens einführen. Damit tritt die vom System vorgegebene Vereinbarung in Kraft, daß Variablennamen, die mit I,J,K,L,M oder N anfangen, Ganzzahlen bezeichnen und alle anderen Namen reelle Zahlen.

Die Komplexität von Ein- und Ausgabe bereitet FORTRAN-Anfängern die meisten Schwierigkeiten. Auch dieses Problem entstand aus der ursprünglichen Verwendung des starren Lochkartenformats. Jeder E/A-Vorgang braucht zwei Befehle: READ oder WRITE bestimmen das E/A-Gerät und die eingesetzten Variablen, während FORMAT Typ und Anordnung der Daten auf der Lochkarte, im Datensatz oder der Eingabezeile des Terminals angibt:

```
READ(1,100) X, I, ICHAR
100 FORMAT(F7.2,I4,A1)
```

READ veranlaßt die Ausführung eines Eingabevorgangs. Die erste Zahl hinter der Klammer bestimmt das Eingabegerät. Die Gerätenummern werden von der Soft- und Hardwareimplementation festgelegt. Die Nummern bezeichnen Terminals, Drucker (nur Ausgabe), Lochkartenleser oder Stanzer und die Dateien der Diskette oder Cassette.

Die zweite Zahl der Klammer bezieht sich auf die Nummer des entsprechenden FORMAT-Befehls. (Beachten Sie, daß FORMAT in unserem Beispiel mit der gleichen Zahl versehen wurde.) Da FORMAT nicht ausgeführt wird, sondern nur Informationen liefert, kann es an einer beliebigen Stelle des Programms stehen. Die Befehlsnummer ist der einzige Bezug, und so können mehrere READs oder WRITEs das gleiche FORMAT einsetzen. Der READ-Klammer folgt die Variablenliste, denen die Eingabewerte zugeordnet werden sollen.

FORMAT enthält eine Beschreibung jeder angegebenen Variable. Hier die wichtigsten:

- F: F7.2 zeigt beispielsweise an, daß die Eingabe für X eine reelle Zahl ist, die ersten sieben Spalten der Zeile, Karte oder des Datensatzes belegt und hinter dem Punkt zwei Ziffern hat.
- I: I4 bestimmt die Eingabe in I als Ganzzahl, die vier Spalten belegt.
- A: A1 legt den Wert der nächsten Spalte als Zeichen fest.

FORTRAN kann Zeichen nur verarbeiten, wenn sie als Ganzzahlvariablen gespeichert werden. Die maximale Zahl pro Variable hängt von der Länge ab, die das System einer Ganzzahlvariablen zuweist. Bei Ganzzahlen im 16-Bit-Format können nur A1 und A2 verwandt werden. In einigen Sprachversionen ist es jedoch auch möglich, dafür Variablen für reelle Zahlen einzusetzen. Da sich Variablen sowohl numerisch als auch alphanumerisch verarbeiten lassen und die Speicherung nicht immer dem ASCII-Code entspricht, kann hier leicht Verwirrung entstehen. So müssen längere Strings z. B. über Ganzzahlenarrays bearbeitet werden.

Die eingegebenen Werte müssen genau den Variablenbeschreibungen entsprechen, da sonst Ablauffehler auftreten. Die meisten modernen Systeme verfügen jedoch über ein „freies Eingabeformat", beispielsweise READ (1,*)A,B,C oder auch READ A,B,C, das sich speziell für die Tastatureingabe eignet, bei der exakte Abstände zwischen den einzelnen Werten nur schwer einzuhalten sind. Dieses freie Eingabeformat arbeitet wie der BASIC-Befehl INPUT. Ausgaben (mit dem WRITE-Befehl) funktionieren ähnlich:

```
WRITE(1,200) A,B,C
200 FORMAT(3F7.2)
```

Beachten Sie, daß Beschreibungen sich mehrfach einsetzen lassen und nicht für jede Variable neu angelegt werden müssen.

Zeichenketten lassen sich mit der speziellen H-Beschreibung ausgeben. Sie ist jedoch recht umständlich, da darin die exakte Zeichenzahl aufgeführt sein muß:

Vergleichende Logik

Logische Operatoren und Verbindungselemente:

FORTRAN	BASIC
.EQ.	=
.NE.	<>
.GT.	>
.GE.	>=
.LT.	<
.LE.	<=
.NOT.	NOT
.AND.	AND
.OR.	OR

Schnelle Zahlen

FORTRAN IV kennt eine Reihe numerischer Datentypen:

- **INTEGER und REAL sind Ganzzahlen und reelle Zahlen**
- **DOUBLE PRECISION sind Fließkommazahlen, die über die doppelte Menge an Speicherwörtern verfügen wie normale reelle Zahlen.**
- **COMPLEX steht für komplexe Zahlen, die einen reellen und einen imaginären Teil besitzen.**
- **LOGICAL entspricht dem PASCAL-Typ BOOLEAN und kann die Werte .TRUE. und .FALSE. annehmen (die Punkte sind notwendig).**
- **Variablennamen können bis zu sechs Zeichen lang sein. Sie dürfen nur alphanumerische Zeichen enthalten. Das erste Zeichen muß ein Buchstabe sein.**
- **Variablen lassen sich nur am Anfang des Programms mit einem Deklarationsbefehl anlegen, etwa:**

```
INTEGER NUM1,NUM2
LOGICAL FNSH
```

```
      WRITE(1,300)ANS
  300 FORMAT(18HDIE ANTWORT
     LAUTET,F7.2)
```

Bei neueren FORTRAN-Versionen lassen sich auch in Anführungsstrichen eingeschlosse Strings einsetzen:

```
      WRITE(1,300)ANS
  300 FORMAT(,DIE ANTWORT
     LAUTET',F7.2)
```

Bei der Ausgabe an einen Drucker setzt FORTRAN das erste gedruckte Zeichen zur Steuerung des Papiers ein. Ein typisches System kann beispielsweise das Leerzeichen für normale Zeichenabstände und die Zeichen 1,2,3 etc. für die gewünschte Anzahl Leerzeilen verwenden. Da diese Eigenheit oft vergessen wird, geht leicht das erste Zeichen der Ausgabe verloren, während der Drucker unvorhersehbare Dinge mit dem Papier anstellt. Hier ein Beispiel für die Druckerausgabe:

```
      WRITE(2,400)I,J,K,L,M,N
      400 FORMAT(1H,2(I6,5X))
```

Beachten Sie das zusätzliche 1H, für die Papiersteuerung. X erzeugt die angegebene Zahl Leerzeichen. Eine Beschreibungsfolge kann geklammert werden – zum Beispiel 2(I6,5X).

Wir verwenden in allen Beispielen die Gerätenummer 1 für die Ein- und Ausgabe per Terminal. Der korrekte Wert hängt allerdings von der eingesetzten Implementation ab.

Sterne statt Pfeil

Das mathematische Format und die Zuordnungen von FORTRAN sind BASIC sehr ähnlich, mit dem einzigen Unterschied, daß FORTRAN für Potenzen statt des Aufwärtspfeils (↑) doppelte Sterne (**) einsetzt. In den mathematischen Ausdrücken lassen sich reelle Zahlen, Ganzzahlen und sogar Werte mit doppelter Präzision vermischen. Dabei ist jedoch Vorsicht geboten, da leicht ein Teil der Berechnung mit der Ganzzahlenarithmetik ausgeführt wird, auch wenn das Ergebnis ein reeller Wert sein soll.

```
I=2
J=3
A=I/J
```

Hier hat A den Wert 0, da die Teilung in der Ganzzahlenarithmetik erfolgte. Die Funktion FLOAT kann die Ganzzahlenwerte jedoch in reelle Zahlen verwandeln:

```
A=FLOAT(I)/FLOAT(J)
```

liefert das korrekte Ergebnis.

Da FORTRAN als mathematische Sprache konzipiert wurde, besitzt sie selbstverständlich viele Standardfunktionen. Die Liste ist jedoch zu lang, um sie hier abdrucken zu können. Sie enthält außer den üblichen triogonometrischen und logarithmischen Funktionen auch viele, die nur für Mathematiker und Ingenieure von Bedeutung sind. Die meisten Funktionen haben eine Version für reelle Zahlen, eine für Zahlen mit doppelter Präzision und, falls nötig, auch noch das Ganzzahlenformat.

Einer der Hauptnachteile von FORTRAN war das Fehlen von Steuerstrukturen. FORTRAN 77 beseitigt dieses Problem jedoch teilweise. So wird der vertraute GOTO-Befehl sehr oft eingesetzt. Er kann jeden mit Nummer versehenen ausführbaren Befehl (nicht aber FORMAT) ansprechen.

Es gibt zwei Variationen des IF-Befehls: Das logische IF entspricht dem IF in BASIC. Es hat das Format:

```
IF (logischer Ausdruck) ausführbarer Befehl
```

Das arithmetische IF wird oft als das schlechteste Beispiel einer Steuerstruktur zitiert. Es hat folgendes Format:

```
IF (arithmetischer Ausdruck) S1, S2, S3
```

wobei S1, S2 und S3 Befehlsnummern sind. Die Steuerung wird an die Befehlsnummer S1 übergeben, wenn der Wert des arithmetischen Ausdrucks kleiner als Null ist, an S2, wenn er gleich Null ist und an S3, wenn er darüber liegt. Für die Programmierung einer dreifachen Verzweigung ist dies sehr praktisch, führt aber zu einem noch erheblich komplizierteren „Spaghetticode" als GOTO.

Durchschnitt

```
C     DAS PROGRAMM LIEST ZAHLEN
C     UND DRUCKT DEN DURCHSCHNITT AUS
C     DER INPUT WIRD DURCH DIE EINGABE
C     EINER
C     NEGATIVEN ZAHL BEENDET
C
C     ZAEHLER UND SUMME INITIALISIEREN
C
      SUMME=0
      IZAEHLER=0
C
C     NAECHSTE ZAHL LESEN
C
100   READ(1,10)X
C
C     TESTEN, OB SIE NEGATIV IST
C
      IF (X) 300,200,200
C
C     POSITIVE ZAHL
C
200   IZAEHLER=IZAEHLER+1
      SUMME=SUMME+X
      GOTO 100
C
C     ENDE DER EINGABE
C
300   AVGE=SUMME/FLOAT(IZAEHLER)
      WRITE(1,20)AVGE
      STOP
C
C     FORMATIEREN
C
10    FORMAT(F9.2)
20    FORMAT (20HDER DURCHSCHNITT
      IST,F9.2)
      END
```

Zahlenakrobatik

Das Array ist zwar die einzige Datenstruktur von FORTRAN, doch erhält die Sprache viel Flexibilität durch den komplexen Einsatz von Subroutinen. Wir untersuchen die speziellen Techniken für die Zahlenverarbeitung mit Hilfe dieser Sprache.

Das Array ist die einzige Datenstruktur, die dem FORTRAN-Programmierer zur Verfügung steht. Wie in BASIC müssen Arrays vor ihrem Einsatz deklariert werden – direkt am Programmanfang, da FORTRAN eine compilierte Sprache ist. Diese Deklaration erfolgt mit dem Befehl DIMENSION, der die gleiche Funktion hat wie DIM in BASIC.

Bei systemvorgegebenen Datentypen (alle Variablennamen, die mit I, J, K, L, M oder N anfangen, stellen Ganzzahlen dar und alle anderen Namen reelle Zahlen) reserviert

```
DIMENSION A(20), I(50)
```

den Speicherplatz für ein Array mit bis zu 20 reellen Zahlen und ein weiteres Array für bis zu 50 Ganzzahlen. Befehle, die die Systemeinstellung verändern (INTEGER, REAL, DOUBLE PRECISION, COMPLEX oder LOGICAL) müssen zusätzlich zu der Anweisung DIMENSION gegeben werden. Arrays lassen sich aber auch gleich in der Typendeklaration definieren:

```
INTEGER ARR1(20)
REAL NUMS(30)
```

Der erste Befehl reserviert Platz für ein Array mit 20 Ganzzahlen und der zweite für 30 reelle Zahlen.

Zweidimensionale Arrays sind möglich, einige Sprachversionen erlauben sogar drei oder mehr Dimensionen. Zweidimensionale Arrays werden auf übliche Weise deklariert:

```
DIMENSION ARR(20,30)
INTEGER ARR(20,30)
```

Jede Ganzzahlkonstante oder Variable läßt sich als Arrayindex einsetzen, beispielsweise ARR(3,4) oder INTARR(I).

FORTRAN IV verarbeitet zwar keine Zeichenstrings, doch stehen bei FORTRAN 77 dafür andere Möglichkeiten zur Verfügung (mit FORTRAN 77 werden wir uns in der nächsten Folge genauer beschäftigen). Zeichen werden wie Ganzzahlen in normalen Ganzzahlvariablen gespeichert. In der letzten Folge haben wir gesehen, wie sich ein oder mehr Zeichen in Ganzzahlvariablen unterbringen lassen. Zwar kommt das FORTRAN-Ganzzahlenarray einem Zeichenstring sehr nahe, doch ist es immer noch recht unbefriedigend, wenn man die Stringverarbeitungsmöglichkeiten von BASIC gewöhnt ist.

Außer den Befehlen IF und GOTO ist die DO-Schleife die wichtigste Steuerstruktur von FORTRAN. Sie funktioniert wie die FOR... NEXT-Schleife in BASIC und hat folgendes Format:

```
DO Sn (intvar) = Startwert, Endwert,
Schrittwert
.....
Sn (letzter Befehl des zu wiederholenden
Blocks)
```

Sn ist die Nummer des letzten Schleifenbefehls. Schleifenzähler (intvar) kann jede Ganzzahlvariable sein. Die Anfangs-, End- und Schrittwerte sind angegeben. In BASIC würde die DO-Schleife so aussehen:

```
FOR intvar = Startwert TO Endwert STEP
Schrittwert
.....
NEXT intvar
```

Folgende Befehle setzen ein Array auf Null:

```
    DO 100 I = 1,20,1
100 ARR(I)=0
```

Der Schrittwert Eins darf weggelassen werden:

```
    DO 100 I = 1,20
100 ARR(I)=0
```

Die numerierte Anweisung, die das Schleifenende kennzeichnet, kann jeder ausführbare Befehl außer IF, GOTO oder DO sein.

Manchmal ist es praktisch, keine feste Schleifenendanweisung zu haben, da sich der Code damit komprimieren läßt. Diese Möglichkeit kann aber auch zu schweren Fehlern führen, und besonders bei verschachtelten Schleifen werden Programme dann völlig unverständlich. Die folgende Initialisierung eines zweidimensionalen Arrays ist jedoch legal:

```
    DO 100 I = 1,20
    DO 100 J = 1,30
100 ARR(I,J)=0
```

(Beachten Sie, daß bei beiden Schleifen die gleiche Endanweisung verwendet wird.)

Oft wird der „Leerbefehl" CONTINUE eingesetzt, um das Schleifenende zu markieren und den Code einfacher zu gestalten. CONTINUE ist ein FORTRAN-Befehl, der keine Wirkung hat und an jeder beliebigen Position eingesetzt werden kann. Jede DO-Schleife muß normalerweise mit einem eigenen CONTINUE abge-

schlossen sein. Das vorige Beispiel sieht dann in ausführlicher Schreibweise so aus:

```
      DO 101 I = 1,20
      DO 100 J = 1,30
      ARR(I,J)=0
  100 CONTINUE
  101 CONTINUE
```

Damit FORTRAN-Programme generell übersichtlicher werden, verlangen einige Programmierer sogar, daß alle DOs, IFs und GOTOs mit einem CONTINUE abgeschlossen werden.

Das speziell für Mathematik ausgelegte FORTRAN besitzt Anweisungen, die besonders gut für den Umgang mit Zahlenarrays geeignet sind. So kann mit einem einzigen READ- oder WRITE-Befehl ein ganzes ein- oder zweidimensionales Array gelesen oder geschrieben werden, wobei FORMAT den Aufbau jeder Zeile oder jedes Datensatzes angibt. Zum Beispiel:

```
      DIMENSION IARR1(20)
      .....
      WRITE(1,10)IARR1
```

veranlaßt die Ausgabe des gesamten Inhalts von IARR1 (20 Ganzzahlen).

10 FORMAT (20I4)

erzeugt 20 vierstellige Ganzzahlen in einer Zeile;

```
      10 FORMAT (I4)
```

erzeugt eine Ganzzahl per Zeile;

```
      10 FORMAT (5I4)
```

Wurzelziehen

```
C     EIN FORTRAN PROGRAMM, DAS SETS VON
C     DREI WERTEN A,B,C, LIEST, DIE DIE
C     KOEFFIZIENTEN DER QUADRATISCHEN
C     GLEICHUNG A*X**2+B*X+C=0 ÜBER DIE
C     ÜBLICHE FORMEL DARSTELLEN
      FUNCTION DISCR(A,B,C)
      DISCR=B*B-4.0*A*C
      RETURN
      END
      SUBROUTINE SOLVE(A,B,C,X1,X2,NROOT)
      D=DISCR(A,B,C)
C     ZWAR LIESSE SICH HIER EIN
C     ARITHMETISCHES IF
C     EINSETZEN, DOCH IST ES NICHT
C     EMPFEHLENSWERT
      IF (D.GE.0.0)GOTO 100
C     DA D UNTER NULL LIEGT, IST DIE WURZEL
C     KOMPLEX
      NROOT=0
      X1=0.0-B/(2.0*A)
      X2=SQR(ABS(D))
      GOTO 300
  100 IF (D.GT.0.0)GOTO 200
C     D IST NULL, DAHER GLEICHE WURZELN
      NROOT=1
      X1=0.0-B/(2.0*A)
      X2=X1
      GOTO 300
C     D IST POSITIV, DAHER ZWEI WURZELN
      NROOT=2
  200 X1=(0.0-B+SQRT(D))/2.0*A
      X2=(0.0-B-SQRT(D))/2.0*A
  300 RETURN
      END
C     ERST DIE ANZAHL DER SETS LESEN
C     READ(2,10)NVAL
C     ÜBERSCHRIFTEN ANZEIGEN
      WRITE(3,11)
      DO 500 L=1, NVAL
C     DREI WERTE LESEN UND PRÜFEN, OB SIE
C     GEEIGNET SIND
  100 READ (1,12) A,B,C
      IF (A.EQ.0.0) GOTO 500
C     DIE WERTE SIND GEEIGNET, DAHER
C     SUBROUTINE AUFRUFEN
      CALL SOLVE(A,B,C,X1,X2, NROOT)
      IF (NROOT.GT.0)GOTO 200
C     KOMPLEXE WURZELN
      WRITE (1,13)A,B,X1,X2
      GOTO 500
  200 IF (NROOT.GT.1)GOTO 300
C     GLEICHE WURZELN
      WRITE(1,14)A,B,C,X1,X2
      GOTO 500
C     ZWEI WURZELN
  300 WRITE(1,15)A,B,C,X1,X2
      GOTO 500
C     UNGÜLTIGE WERTE FÜR A, B UND C
  400 WRITE(1,16)A,B,C
  500 CONTINUE
      STOP
C     FORMAT BEFEHLE
   10 FORMAT (I4)
   11 FORMAT(1H ,8X,1HA,8X,1HB,8X,1HC,8X,
     4HTYPE,8X,2HX1,8X,2HX2)
   12 FORMAT(1H ,3F7.2)
   13 FORMAT(1H ,4X,3(F7.2,2X),4X,7HCOMPLEX,
     X,F7.2,3X,F7.2)
   14 FORMAT(1H ,4X,3(F7.2,2X),4X,5HEQUAL3X,
     F7.2,3XF7.2)
   15 FORMAT(1H ,4X,3,(F.2.2X),4X,
     7HUNEQUAL,X,
     F7.2.3X,F7.2)
   16 FORMAT(1H ,4X,3,(F7.2,2X),4X,7HINVALID)
      END
```

Durchschnittsberechnung

```
C     EINE FORTRAN-FUNKTION, DIE
C     DEN DURCHSCHNITT
C     (IN REELLEN ZAHLEN) EINES STETS MIT N
C     GANZZAHLEN
C     DES GANZZAHLENARRAYS IARR BERECHNET
      FUNCTION AVGE (IARR,N)
      DIMENSION IARR (N)
      SUM=0
      DO 100 I=1,N
      SUM=SUM+FLOAT (IARR(I))
  100 CONTINUE
      AVGE=SUM/FLOAT (N)
      RETURN
      END
```

vier Zeilen mit je fünf Zahlen und

```
10 FORMAT (20A1)
```

einen String von 20 Zeichen in einer Zeile.

Beim Einlesen oder Ausgeben von zweidimensionalen Arrays entsteht oft ein Problem, da FORTRAN, im Gegensatz zu anderen Sprachen, zuerst die Spalten und dann erst die Zeilen speichert. Wenn Sie hier nicht aufpassen, entsteht eine versetzte Version des Arrays. Es gibt jedoch eine spezielle Form der DO-Schleife, die „implizierte Schleifenanweisung", die sich gut für die Arbeit mit einem Teilarray eignet. Implizierte Schleifenanweisungen werden mit dem Variablennamen in die Befehle READ oder WRITE eingefügt. Die folgenden Zeilen füllen beispielsweise das Stringarray IARR2(10,20) mit zehn Strings zu je 20 Zeichen:

```
    DO 100 I=1,10
    READ (1,10) (IARR2(I,J), J=1,20)
10  FORMAT (20A1)
100 CONTINUE
```

FORTRAN bietet die Möglichkeit, mit Subroutinen zu arbeiten – wenn auch mit einigen Einschränkungen. Die Subroutinen sind völlig unabhängig und arbeiten hauptsächlich mit lokalen Variablen. Sie werden entweder an das Ende des Hauptprogramms angehängt oder separat geschrieben und compiliert. Subroutinen können durchaus die gleichen Zeilennummern einsetzen, die auch das Hauptprogramm verwendet (in den meisten Fällen sogar die gleichen Variablennamen). Einfache Parameter werden durch Wertaufruf übergeben, wobei Werte, die von der Subroutine geändert wurden, auch die entsprechenden Variablen des aufrufenden Programms beeinflussen. Globale Variablen, auf die sich Subprogramme und Hauptprogramm beziehen, müssen in beiden Programmodulen mit COMMON deklariert werden. Es lassen sich eine ganze Reihe von COMMON-Blöcken anlegen und mit Namen versehen, doch meistens genügt ein einziger Block dafür.

Die Variablen der COMMON-Anweisungen müssen einander in Typ und Gesamtlänge entsprechen. So könnte das Hauptprogramm beispielsweise den Befehl

```
COMMON I1,I2,I3
```

enthalten und die Subroutine

```
COMMON I(3)
```

In beiden Fällen wird der Speicherplatz auf die gleiche Weise eingesetzt, wobei die Parameter sich im ersten Fall auf Ganzzahlenvariablen beziehen und im zweiten Fall auf drei Arrayelemente des gleichen Typs.

Ein typischer Subroutinenaufruf sieht folgendermaßen aus:

```
CALL SUBNAME(X,Y,I,J)
```

wobei die Subroutine mit dem Befehl

```
SUBROUTINE SUBNAME(A,B,L,M)
```

anfängt. (Beachten Sie, daß die Parameter im Inneren der Klammern in Bezug auf Anzahl und Typ übereinstimmen.)

Die Anweisung RETURN gibt die Steuerung an das aufrufende Programm zurück (in einer Subroutine können sich mehrere RETURNs befinden). Eines der RETURNs sollte jedoch in jedem Fall ausgeführt werden – unabhängig von dem Weg, der durch die Subroutine führt. In Subroutinen wie im Hauptprogramm muß der letzte Befehl immer ein END sein.

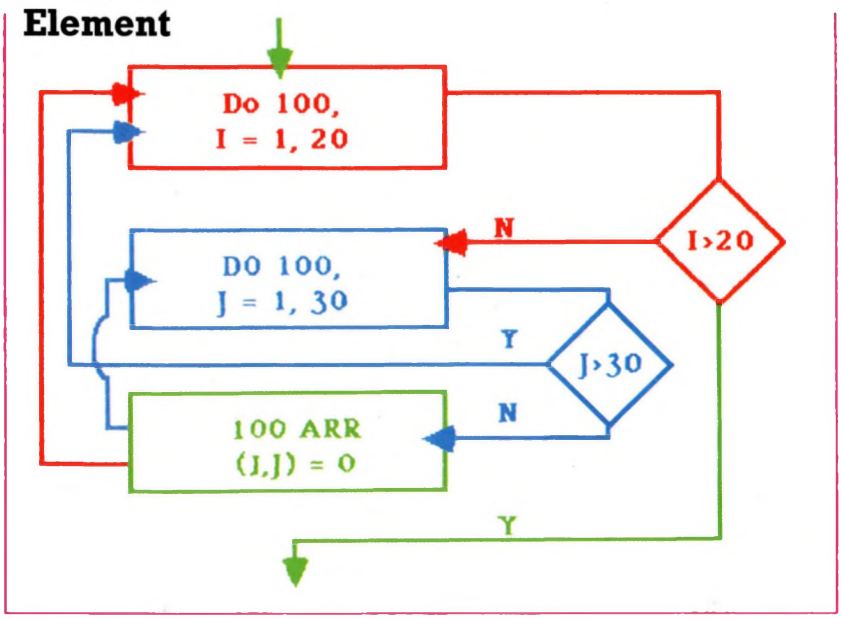
Bei der Parameterübergabe ist es in bestimmter Hinsicht möglich, die Größe eines Arrays dynamisch zu verändern. Obwohl das in einer interpretierten Sprache wie BASIC leicht auszuführen ist, gibt es diese Möglichkeit in compilierten Sprachen nur selten, da der Speicherplatz bei der Compilierung fest zugeordnet wird und nicht erst mit Geschwindigkeitsverlust beim Ablauf des Programms.

Da Subroutinen nur einen einzigen Wert zurückliefern, können sie auch (wie in PASCAL) als FUNCTION angelegt werden. Derartige Subprogramme werden, wie normale Subroutinen, separat geschrieben, beginnen aber mit der Anweisung FUNCTION. Vor dem RETURN sollte eine Zuweisung auf den Funktionsnamen liegen. Der Funktionsname selbst bestimmt den von der Funktion zurückgelieferten Datentyp, wobei die Vorgaben der vom System vorgegebenen Variablennamen angewandt werden. Diese Anforderung läßt sich zwar umgehen, sollte aber wenn möglich beibehalten werden. Das Hauptprogramm setzt die definierten Funktionen auf die gleiche Weise ein wie die normalen Systemfunktionen. Der Funktionsname (mit zugehöriger Parameterliste) kann dabei selbstverständlich in jedem Ausdruck erscheinen, in dem eine Variable dieses Typs auftreten darf.

Da FORTRAN keinen fest definierten Befehl für das Schleifenende besitzt, können mehrere Schleifen mit der gleichen Endanweisung abgeschlossen werden. Damit läßt sich sehr kompakter Programmcode schreiben (siehe Bild). In BASIC ist der gleiche Ablauf zwar weniger komprimiert, aber dafür auch weit leichter zu lesen.

```
100 FOR I=1 TO 20
110 FOR J=1 TO 30
120 LET A(I,J)=0
130 NEXT J
140 NEXT I
```

Das letzte Element



Aus alt mach neu

Da FORTRAN aus den fünfziger Jahren stammt, fehlen dieser Sprache viele Strukturen und Möglichkeiten, die in modernen Programmiersprachen als selbstverständlich angesehen werden. Wir sehen uns einige Techniken an, die diesen Mangel beheben.

FORTRAN fehlen zwei wichtige Eigenschaften: Es hat keine brauchbaren Steuerstrukturen, die die Sprache leicht verständlich machen und verfügt außerdem nur über eingeschränkte Möglichkeiten der Zeichenverarbeitung. Im Laufe der Zeit gab es eine Reihe von Versuchen, diese Einschränkungen gänzlich zu beseitigen.

Eine Möglichkeit dafür bietet der Einsatz eines „Vorprozessors" (eine Art Compiler auf einer höheren Ebene). Er bearbeitet Programme, die mit einer erweiterten FORTRAN-Version erstellt wurden und setzt für alle Zusatzeigenschaften die entsprechenden Abläufe des Standard-FORTRAN ein. Damit lassen sich strukturierte Programme schreiben, die in kompilierter Form dem Standard entsprechen und sich allgemein verwenden lassen. WATFOR und RATFOR sind die am weitest verbreiteten Versionen.

Die Eigenschaften von FORTRAN 77 entsprechen denen der bekannteren Vorprozessoren. Aus diesem Grund blieb die Essenz der Sprache erhalten, obwohl viele Compiler für Microcomputer auf dem erweiterten FORTRAN IV aufbauen und nicht auf FORTRAN 77.

FORTRAN besitzt keine Blockstruktur wie ALGOL, PASCAL oder C, was sich speziell bei der Einrichtung von Steuerstrukturen negativ bemerkbar macht. Außerdem können nicht mehrere Befehle zu einer Anweisung zusammengefaßt werden (beispielsweise durch BEGIN...END). Blöcke lassen sich nur isolieren, wenn sie als separate Subroutinen angelegt werden, oder von GOTO-Befehlen umgeben sind. Es gibt jedoch eine neue Steuerstruktur, die allerdings nicht so recht zu der Sprache paßt: IF...THEN...ELSE...ENDIF:

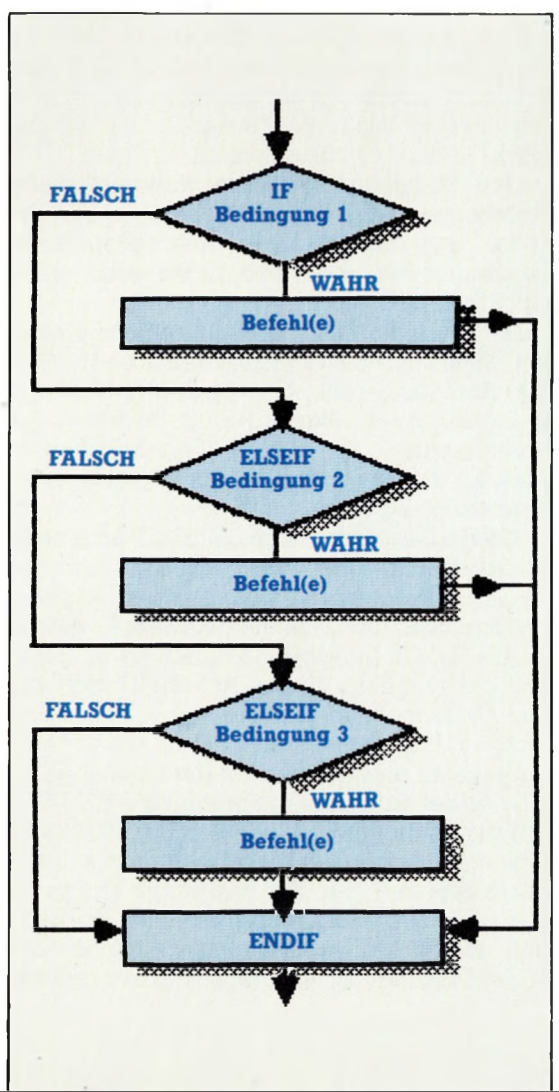
```
IF (logischer Ausdruck) THEN
    .....
    Anweisungen
    .....
ELSE
    .....
    Anweisungen
    .....
ENDIF
```

Der logische Ausdruck hat das gleiche Format wie Standard-FORTRAN. Normalerweise wird der ELSE-Teil weggelassen.

Eine sehr praktische Form dieser Anweisung arbeitet mit verschachtelten IFs:

```
IF (logischer Ausdruck) THEN
    .....
    Anweisungen
    .....
ELSEIF
    .....
    Anweisungen
    .....
ELSEIF
    .....
ELSE
    .....
ENDIF
```

Es lassen sich beliebig viele ELSEIFs für tiefere Strukturierung verwenden.



In den meisten BASIC-Versionen sind verschachtelte IFs möglich. FORTRAN hat jedoch zusätzlich die Struktur ELSEIF, mit der sich IF-Anweisungen über mehrere Programmzeilen bis zu einer beliebigen Tiefe verschachteln lassen. ENDIF beendet die Struktur.

Die zweite wesentliche Verbesserung ist die Einführung eines Datentyps für Zeichen. Zeichenstrings werden – wie die anderen Deklarationen – am Programmanfang in einem der folgenden Formate festgelegt:

```
CHARACTER*9          CH1,CH2
CHARACTER            CH3*7,CH4*16
```

Die maximale Länge eines Strings wird hier mit *n angegeben, wobei n eine Ganzzahl ist. Dieser Vorgang kann entweder auf das Schlüsselwort CHARACTER angewandt werden (wenn alle Strings die gleiche Länge haben), oder auf einzelne Strings mit individuellen Bezeichnungen. Die beiden aufgeführten Deklarationen ergeben zwei Strings der Länge 9, einen mit Länge 7 und einen mit Länge 16.

Stringarrays werden wie üblich definiert:

```
CHARACTER*4 CHARS(50)
```

deklariert ein Array von 50 Strings mit jeweils vier Zeichen. Stringkonstanten werden Stringvariablen folgendermaßen zugeordnet:

```
CH1'ABCDEFGHI'
```

Ist der zugeordnete String zu kurz, werden Leerzeichen angehängt; ist er zu lang, werden die Zeichen am Stringende ignoriert.

Strings können mit normalen relationalen

CODEKNACKER

Dieses FORTRAN-Programm liest Daten aus einer Datei namens GEHEIM und dechiffriert über einen Decodierschlüssel den chiffrierten Text.

```
C     EIN PROGRAMM, DAS GEHEIMNACH-
C     RICHTEN DECODIERT. BEACHTEN SIE DIE
C     ANWEISUNG ‚PROGRAM' IN FORTRAN 77
C
      INTEGER COUNT,PTRIN,PTROUT
      LOGICAL FILE
      CHARACTER *10 NUM, CHAR
      CHARACTER *30 IN, OUT
      DATA COUNT /0/
C
C     ‚DATA' ANWEISUNG ZUR INITIALISIERUNG
C     EINER VARIABLE
C
C     WENN KEINE GEHEIMDATEI EXISTIERT,
C     FEHLERMELDUNG AUSGEBEN
C
      INQUIRE (FILE='GEHEIM',EXIST=FILE)
      IF (.NOT.FILE) THEN
        WRITE(1,10)
        STOP
      ENDIF
C
C     DATEIEN ERÖFFNEN
C
      OPEN (UNIT=12,FILE='MELDUNG',STATUS
        ='NEW')
C
C     DECODIERSCHLÜSSEL ANGEBEN
C
      NUM='0123456789'
      CHAR='MW3 OD$%^T'
C
C     GEHEIME MELDUNG LESEN UND
C     DECODIEREN
C
100   READ (11,20,END=1000)IN
        OUT=' '
        PTROUT=1
        DO 200I=1,30
          PTRIN=INDEX (CHAR,IN(I:I))
C
C     STANDARDFUNKTION 'INDEX'
C     BESTIMMT DIE POSITION
C     DES TEILSTRING IN(I:I) IM STRING CHAR
        IF (PTRIN.NE.0) THEN
          OUT (PTROUT:PTROUT)=NUM
          (PTRIN:PTRIN)
          PTROUT=PTROUT+1
        ENDIF
200   CONTINUE
      WRITE (12,20) OUT
C
C     BEACHTEN SIE, DASS EIN- UND AUSGABE
C     DAS GLEICHE FORMAT HABEN
C
        COUNT=COUNT+1
      GOTO 100
C
C     ENDVORGANG
C     MIT 'ENDFILE' WIRD DIE MARKIERUNG
C     DES DATEIENDES GESCHRIEBEN
C
1000  ENDFILE (UNIT=12)
      STOP
C
C     FORMATS
10    FORMAT (1H,'ES EXISTIERT
      KEINE GEHEIMDATEI')
20    FORMAT (A)
C
C     BEACHTEN SIE, DASS NICHT
C     ANGEGEBEN WERDEN MUSS,
C     WIEVIEL ZEICHEN
C     EINZUGEBEN SIND
C
      END
```

Operatoren wie GT.,LT etc. verglichen werden. Es gibt jedoch zwei neue Operatoren nur für Stringfunktionen. Der Teilstringoperator liefert eine Zeichenfolge des Strings.

```
CH1(3:5)
```

holt den Teilstring von CH1, der beim dritten Zeichen beginnt und beim fünften endet.

```
CH1='ABCDEFGHI'
CH2=CH1(3:5)
```

legt in CH2 den Inhalt ‚CDE', oder genauer 'CDE ' ab. Beachten Sie, daß die zweite Zahl die Position des letzten Zeichens angibt und nicht die Länge des Teilstrings.

Der Verkettungs-Operator (//) verbindet zwei Strings miteinander.

```
CHARACTER CH1*4,CH2*4,CH3*8
CH1='ABCD'
CH2='WXYZ'
CH3=CH1//CH2
```

ergibt CH3 mit dem Inhalt ‚ABCDWXYZ'.

Um FORTRAN richtig nutzen zu können, ist ein System mit Diskettenlaufwerken nötig. Die meisten FORTRAN-Versionen können sequentiell und direkt auf Dateien zugreifen. Die Quelle oder die Bestimmung eines E/A-Vorgangs wird von der Gerätebezeichnung des READ- oder WRITE-Befehls bestimmt. Einige Parameterwerte – normalerweise die niedrigeren – sind für E/A-Geräte wie Tastatur, Bildschirm und Drucker reserviert, die anderen Nummern stehen dem Programmierer zur Verfügung. Der Befehl OPEN ordnet einer Diskettendatei eine Namen zu:

```
OPEN (UNIT=Ganzzahlausdruck,
FILE=Dateiname, STATUS=Status)
```

Der Ganzzahlenwert für UNIT bezeichnet in den folgenden READ- oder WRITE-Befehlen die Dateinummer. Status kann mehrere Werte annehmen – beispielsweise OLD für eine Eingabedatei, die es bereits gibt, oder NEW für eine Ausgabedatei, die angelegt werden soll.

Falls die angesprochene Datei vom einfachen sequentiellen Format abweicht, kann OPEN um eine Reihe von Zusatzangaben ergänzt werden: ACCESS= bestimmt den direkten oder sequentiellen Zugriff; FORM= gibt an, ob die Datei formatiert oder unformatiert ist; IOSTAT bietet die Möglichkeit, Fehler abzufangen, falls keine Dateizuordnung zustande kommt; RECL= legt die Dateilänge fest.

CLOSE hat einen ähnlichen Aufbau:

```
CLOSE (UNIT=Ganzzahlausdruck,
STATUS=Status)
```

Der Befehl schließt eröffnete Dateien, wird aber nur selten benötigt, da bei Beendigung des Programms alle Dateien automatisch geschlossen werden.

Von den anderen Dateibefehlen ist INQUIRE möglicherweise am interessantesten, da sich damit der aktuelle Status jeder beliebigen Datei feststellen läßt.

Auch zu READ und WRITE sind Zusätze möglich, die die E/A-Vorgänge bestimmen.

```
READ(7,20,REC=I)A,B,C
```

liest beispielsweise Record I einer Datei, die für den direkten Zugriff eröffnet wurde.

```
READ(7,11,END=1000)A,B,C
```

übergibt die Steuerung an Anweisung Nummer 1000, wenn in einer sequentiellen Datei das Dateiende gefunden wird.

Zum Schluß noch ein Blick auf die FORTRAN-Compiler. Sie arbeiten schnell und effektiv und erzeugen schnellen, kompakten und leistungsfähigen Code, der besonders bei Echtzeitanwendungen große Vorteile bringt.

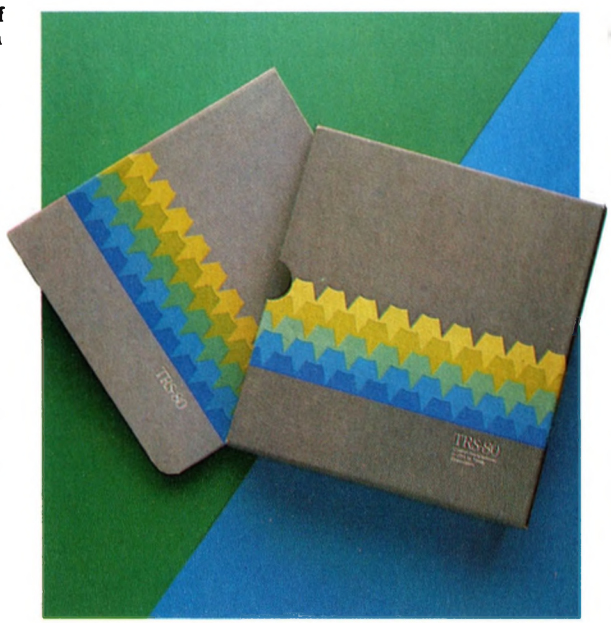
Fehler lassen sich mit den FORTRAN-Compilern jedoch nicht gut abfangen. So akzeptieren die Compiler Fehler, die beispielsweise vor einem PASCAL-Compiler nie bestehen könnten. So haben Leerzeichen in einer FORTRAN-Anweisung etwa keine Bedeutung. Ein Compiler nimmt daher oft als erstes alle Leerzeichen aus einer Befehlszeile. Nun legt der Befehl

```
DO 100 I=1,100
```

eine Schleife an, die 100 mal ausgeführt wird. Ein Punkt anstelle des Kommas ergibt nach Beseitigung aller Leerzeichen jedoch

```
DO100I=1.100
```

und damit eine völlig legitime Wertzuweisung für die Variable DO100I. Ein Gerücht besagt, daß dieser kleine Irrtum im Weltraumprogramm der Vereinigten Staaten zu einem sehr teueren Fehler geführt haben soll. FORTRAN war lange Zeit die Standard Programmiersprache der Streitkräfte. Fehler dieser Art waren Anlaß für die Entwicklung von ADA.



Da FORTRAN-Compiler recht teuer sind, gibt es sie normalerweise nur für kommerzielle Maschinen. Das im Bild gezeigte Paket läuft auf dem Tandy 2000. Durch die begrenzten Speicherkapazitäten sind FORTRAN-Anwendungen auf Micros problematisch, wenn nicht unmöglich.

Mit der wachsenden Verfügbarkeit von CP/M auf Heimcomputern wie dem Commodore 128, Schneider CPC und anderen, stehen vielen Anwendern nun auch FORTRAN-Versionen zur Verfügung. Für Besitzer von Schneider Geräten gibt es eine besonders preiswerte Version, das „Nevada FORTRAN". Sie kann bei New Star Software Ltd., 45 Plovers Mead, Wyatts Green, Essex, CM15 OPS bestellt werden.